汇编语言程序设计

新型计算机研究所

张兴军

西一楼B段811室

Tel: 8266 8478 ext. 806

Email: xjzhang@xjtu.edu.cn

计算机语言的发展

- 机器语言（*二进制编码*）
- 汇编语言（*符号式*）
- 高级语言
  - 算法语言（*接近自然语言及面向过程*）
  - 非过程化语言（*面向对象*）
  - 智能性语言（*具有一定智能，抽象问题求解*）

什么是汇编语言？

- 机器语言：机器指令的集合。以二进制形式的指令组成的指令集合，它是计算机惟一能够直接识别和处理的语言

  例如：1000100111011000； 将寄存器BX的内容送到寄存器AX。

  缺点：编写、阅读、改错很不方便

  机器语言描述的程序称为目标程序（可执行程序），只有目标程序CPU才能直接执行

- 汇编语言：机器语言的符号化表示。面向机器的语言，用简单且容易记忆的符号（助记符号）来代替机器语言中“0”、“1”的一种程序设计语言

  例如：

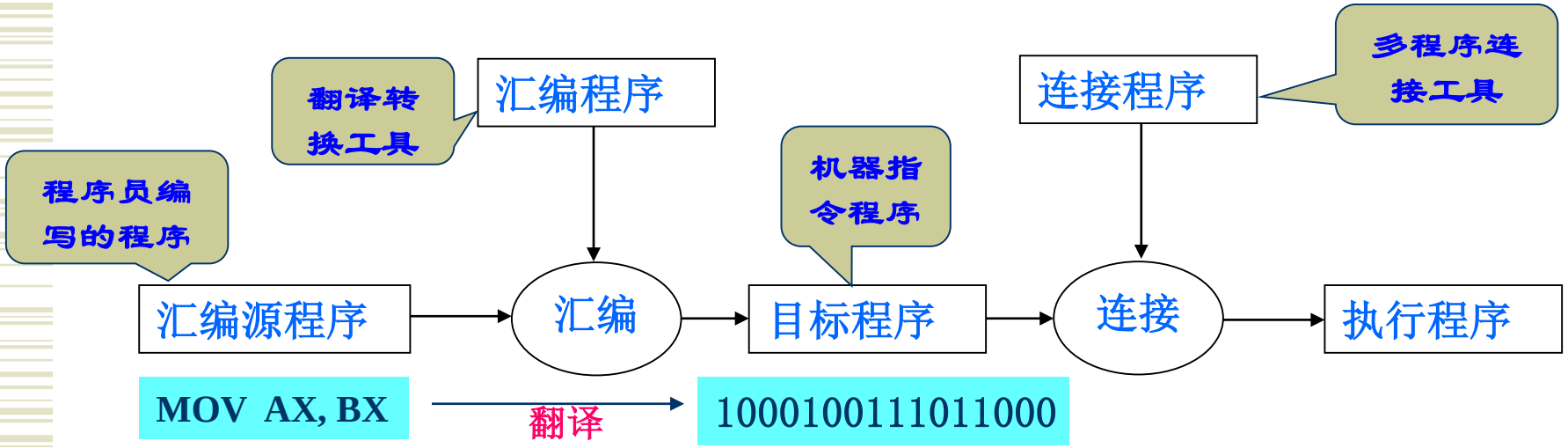
机器语言指令	汇编语言指令
1000100111011000	MOV AX,BX

  汇编语言的执行语句与机器语言的指令是一对一的关系

  现代计算机中使用汇编语言主要是为了某些情况下直接控制计算机硬件以获得高级语言无法实现的功能和性能

用汇编语言编写程序执行过程

汇编语言建立在机器语言之上，由于计算机不能够直接识别这种符号语言，所以汇编语言编写的源程序必须用汇编程序直接翻译成机器语言后才能执行



为什么要学习汇编语言？

- 理解计算机的基本系统结构
- 能够学习到处理器是如何工作的
- 探究数据和指令的内部表述
- 能够创建小巧有效的程序
- 允许程序员绕过高层语言的限制编程
- 有些工作必须用汇编语言完成

编写嵌入式程序、实时应用、编写驱动程序等

汇编语言程序设计与后续课程的关系

- 硬件课程：
  - 微机原理与接口技术：使用汇编语言编写硬件驱动控制程序等
  - 计算机组成原理：如何用硬件实现机器指令，即汇编语言指令功能
  - 计算机系统结构：CPU内部结构、机器指令、存储系统和IO系统优化设计、评价方法
- 软件课程：
  - 编译原理：如何将 高级语言程序模块 通过优化编译转换成 机器指令程序模块
  - 操作系统：基于汇编语言设计实现对计算机系统资源管理、任务管理
  - 高级语言程序设计如何使用汇编语言程序对低层的直接控制

使用教材及其他

- 教材、主要参考书籍
  - 教材：《80X86汇编语言程序设计》，沈美明，温冬婵 编著，清华大学出版社
  - 主要参考书籍：
    - 《汇编语言》（第2版），王爽著，清华大学出版社
    - 《Intel汇编语言程序设计》，Kip R.Irvine 著，温玉杰 等译，电子工业出版社
    - 《IBM PC Assembly Language and Programming 》, Peter Abel, (影印版，清华大学出版社)
- FTP课件下载
  - IP地址：202.117.15.199；用户名、密码：assemble
  - 公共邮箱：huibianyy@163.com；密码：huibian
- 答疑
  - 时间： 上班时间
  - 地点： 西一楼B段811房间 （张兴军），西一楼B段413房间 （董小社）

课程计划安排

- 课内基础知识讲授：32学时
  - 第一章：基础知识 …… 2学时
  - 第二章：80x86计算机组织结构 … 4学时
  - 第三章：指令系统 …… 8学时
  - 第四章：汇编语言程序格式 …… 4学时
  - 第五章：循环与分支程序设计 …… 3学时
  - 第六章：子程序结构 …… 3学时
  - 第七章：高级汇编语言技术 …… 4学时
  - 第八章：输入输出程序设计 …… 2学时
  - 第九章： BIOS和DOS中断 …… 2学时
- 课内上机实验：16学时
  - 第1次：第4周星期六(3月24日) 18:00-22:00
  - 第2次：第5周星期六(3月31日) 18:00-22:00
  - 第3次：第8周星期四(4月19日) 18:00-22:00
  - 第4次：第9周星期四(4月26日) 18:00-22:00

考核方式：

考试成绩 80%

平时作业 5%

平时表现 5%

上机成绩 10%

本课程学习方法

1. 熟练掌握计算机硬件组织结构、功能和工作原理
   - 80X86处理器：寄存器、运算器、PSW 等工作原理
   - 存储器（内存）：数据存储和访问方式
   - 输入/输出：工作原理和控制方式
2. 熟记常用指令、DOS功能中断调用、数据表示
3. 掌握非常用指令、数据表示方式
4. 学习汇编语言程序设计方法
5. 在实践中总结、创新编程优化技巧

注：汇编语言程序设计课程要求掌握如何使用硬件资源；

计算机组成原理课程要求掌握如何设计实现硬件；

计算机系统结构课程要求掌握硬件优化设计和评价方法

第1章　基础知识

1.1 进位计数制与不同基数的数之间的转换

1.2 二进制数和十六进制数运算

1.3 计算机中数和字符的表示

1.4 几种基本的逻辑运算

本章目标

- **掌握**
  - 常用的各种进制数的表示和运算
  - 常用的各种进制数的数制之间的转换规则
  - 带符号数的补码表示方法和补码的运算
- **熟悉**
  - 符号扩展的概念
  - 数据的表数范围
- **了解**
  - 了解计算机存取信息的基本数据类型
  - 了解计算机中字符的表示

1.1 计算机中的数

1. 汇编程序中常用的数制表示

二进制数：计算机硬件唯一识别和使用的数制

以2为基的数制表示法，数由2个数字构成（0、1），二进制数后缀为B，如10110111B。

十进制数：人类自然语言中常用的数制

以10为基的数制表示法，数由10个数字构成（0~9），十进制数后缀为D，如1945D。

十六进制数：程序设计中方便使用和转换的数制

以16为基的数制表示法，数由16个数字构成[0~9、A（10）、B（11）、C（12）、D（13）、E（14）、F（15）]，十六进制数后缀为H，如18ADH。

十进制数的特点

(1) 基数为10：有10个不同的数字符号

0、1、2、3、4、5、6、7、8、9

(2) 逢10进位/借1当10：由于十进制数是逢10进位的，因此同一个数字符号在不同的位置（数的排列先后）代表的数值是不同的

十进制数的特点

(3) 一般表达式

$$D = D_{n-1} \cdot 10^{n-1} + D_{n-2} \cdot 10^{n-2} + \cdots + D_1 \cdot 10^1 + D_0 \cdot 10^0 + D_{-1} \cdot 10^{-1}$$
$$+ D_{-2} \cdot 10^{-2} + \cdots D_{-m} \cdot 10^{-m}$$
$$= \sum_{i=-m}^{n-1} D_i \cdot 10^i$$

在此处键入公式。例如: **666.66D**

- 小数点左边第1位数“6”位于个位，它的值就是6本身
- 小数点左边第2位数“6”位于十位，它的值就是6×10=60
- 小数点左边第3位数“6”位于百位，它的值就是6×100=600
- 小数点右面第1位数“6”位于十分位，它的值是$6\times10^{-1}=0.6$，小数点右面第2位数“6”位于百分位，它的值是$6\times10^{-2}=0.06$，……

二进制数的特点

特点：

(1)基数是2：具有两个不同的基本符号0、1

(2)逢2进1，借1当2

二进制一般表达式：

$$B_{n-1}\cdot 2^{n-1}+B_{n-2}\cdot 2^{n-2}+\cdots+B_1\cdot 2^1+B_0\cdot 2^0+B_{-1}\cdot 2^{-1}$$
$$+B_{-2}\cdot 2^{-2}+\cdots B_{-m}\cdot 2^{-m}$$

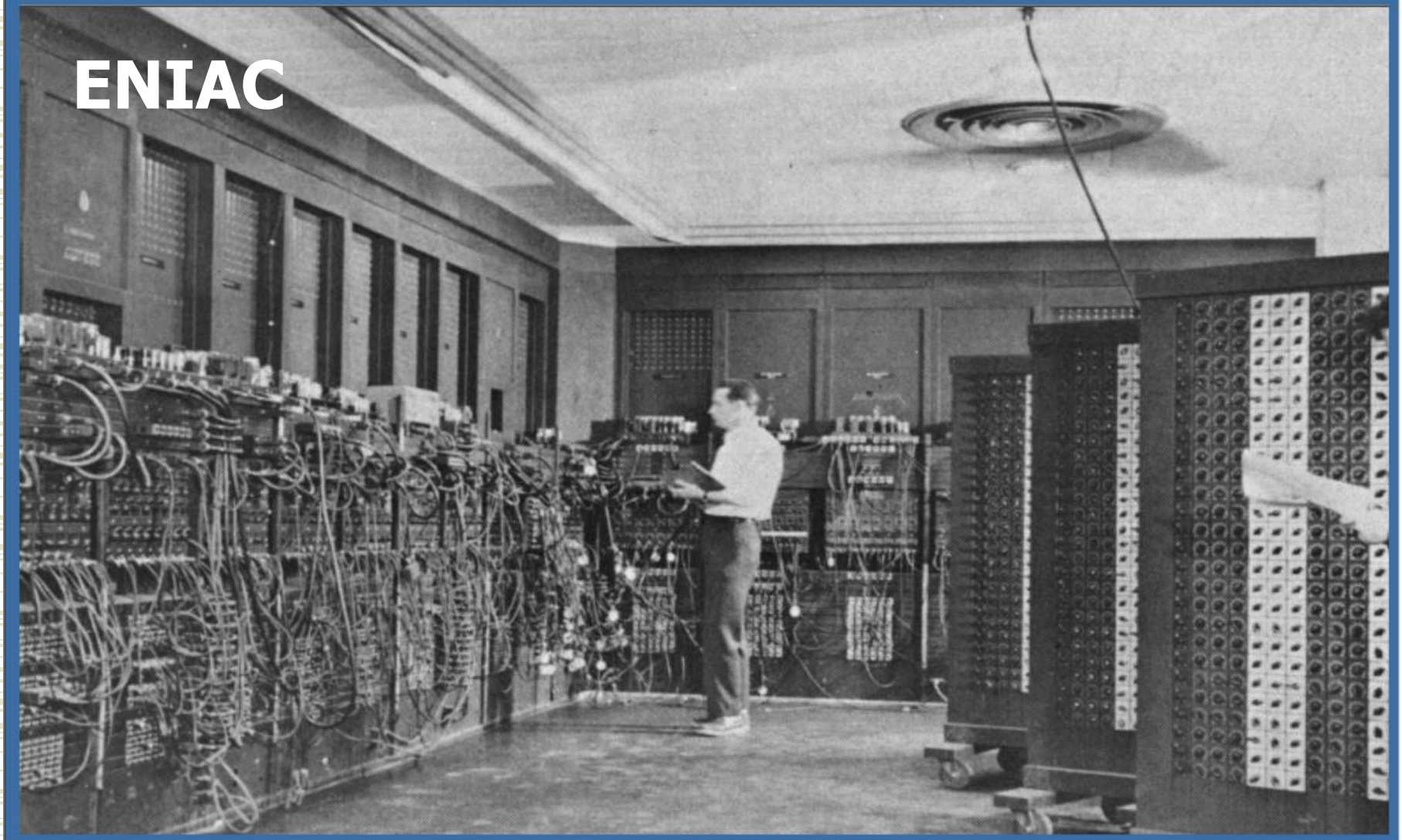
二进制与十进制关系

$$D=\sum_{i=-m}^{n-1}B_i\cdot 2^i$$

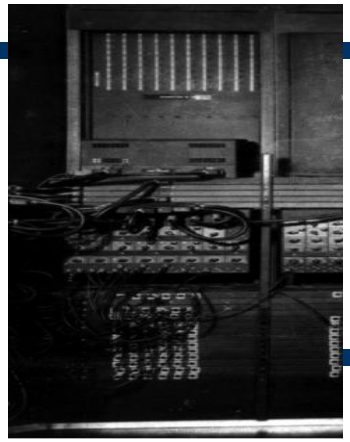
为什么计算机中采用二进制数

- 便于存储及计算的物理实现：二进制数各位上的数码只有0和1两种取值，用计算机内部的电路实现时（高电压、低电压），比十进制数要方便的多
- 工作稳定可靠：抗干扰能力强
- 实际计算机电子线路中
  - 例如：用+5V或+3.3V表示1，用0V表示0
- 若干位排列起来就可表示一个二进制数

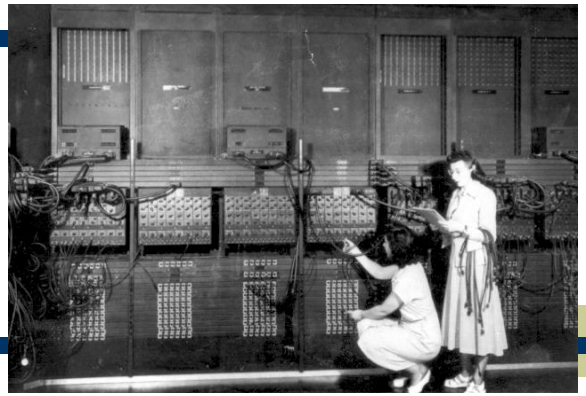
the Modern Computer Age
1946



Decade Counter



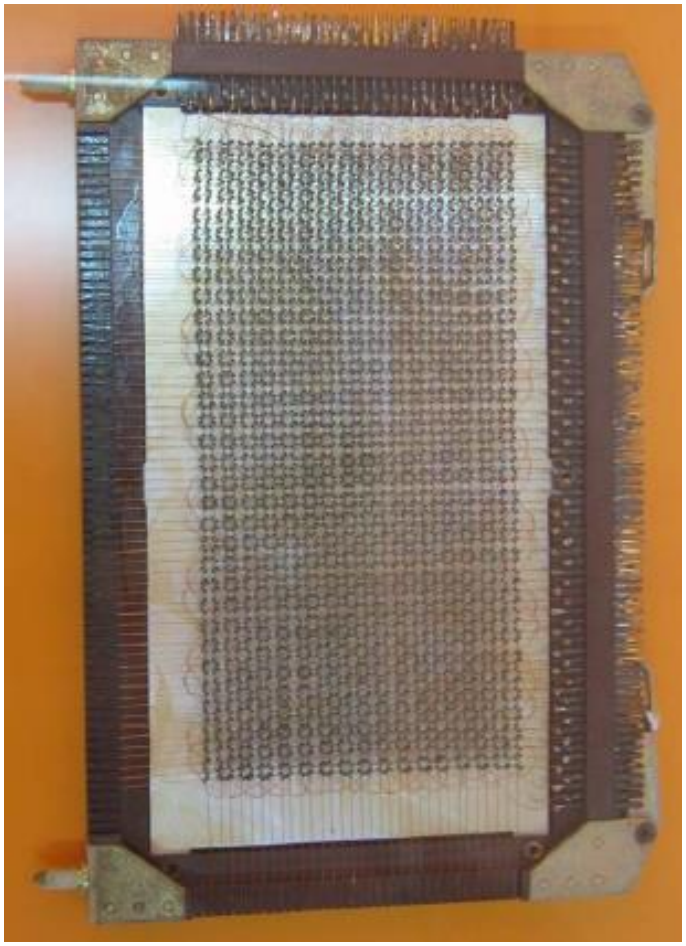
Accumulator



“Wiring” a Program

- 18000 tubes
- 1500 relays
- 174 KW
- 30 tons
- 1800 sq. ft. footprint
- Clock: 100 kHz
- IO: punched cards

早期计算机部件



早期计算机IO

二进制计量单位

- ***比特***：bit，或称位元，简称位，0或1；以“b”表示；最小单位
- ***字节***：byte，位组，8个bit；以“B”表示，一个字符用一个字节表示
- ***字***：word，一般情况下指2个字节
- 2^8=256，2^{10}=1024=1K，2^{16}=64K，2^{20}=1M，2^{24}=16M，2^{30}=1G，2^{32}=4G，…
- 1KB=1024B，1MB=1024KB，1GB=1024MB；1TB=1024GB

几种常用进位记数制的基数和数码

数　　制	基 数	各位数码表示
二进制 Binary	2	0,1
八进制 Octal	8	0,1,2,3,4,5,6,7
十进制 Decimal	10	0,1,2,3,4,5,6,7,8,9
十六进制 Hexadecimal	16	0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F

十进制数 、二进制数、八进制数和十六进制数之间的对应关系

十进制(D)	二进制(B)	八进制(O)	十六进制(H)
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

为什么需要十六进制数

- 便于编程与阅读：二进制的阅读、书写和记忆不方便
  - 用十进制数？
    - 十进制与二进制无直接对应关系，转换困难
    - 如果不转换，用BCD码(Binary Code Decimal，二进制编码的十进制数）表数效率太低，处理困难
      - 0000～1001分别表示8421 BCD码的0～9
      - 1010～1111没有用，浪费6个编码，即37.5%
  - 2^n作为基数的数制转换方便、处理方便，表数效率最高
    - 八进制(000～111)、十六进制(0000～1111)
- 十六进制数与计算机中数据存储、处理的单位长度适用
  - 基本单位：一个二进制位（bit）
  - 常用字符单位：8位二进制数组成的一个字节（byte）
  - 计算机字长：字节的整数倍，8位、16位（字，Word）、32位、64位二进制数
  - 1位十六进制数对应4位二进制数
  - 1位八进制数对应3位二进制数

2. 不同进位计数制之间的数据转换

- **CPU处理的是二进制数**
  - 写程序时，需要把十进制、二进制数转换为十六进制数
  - 阅读程序时，需要将二进制、十六进制转换为十进制数

- **数据转换：**把一种进制数转换为另一种进制的数，其实质是进行基数的转换。基数转换是依据两个有理数相等，其整数部分与小数部分分别相等的原则。

- **转换方法：**转换时，其整数部分与小数部分应分别进行转换，将转换后的结果合并，整数部分与小数部分之间用小数点隔开，就得到相应的转换结果。

(1) 二进制数转换为十进制数的转换规则是“按权值相加”。也就是说，只要把二进制数中数位是“1”的那些位的权值相加，其和就是等效的十进制数。

$$D = \sum_{i=-m}^{n-1} B_i \cdot 2^i$$

$$1101B = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 13D$$

(2) 十六进制转换为十进制数的转换规则

$$D = \sum_{i=-m}^{n-1} h_i \cdot 16^i$$

(2) 十进制数转换为二进制数

整数和小数部分分别进行转换，转换结束后将整数转换结果写在左边，小数转换结果写在右边，中间点上小数点。

- 两种基本方法：降幂法、除法

十进制数转换为二进制数

方法1：降幂法

$$D = B_{n-1} \cdot 2^{n-1} + B_{n-2} \cdot 2^{n-2} + \cdots + B_1 \cdot 2^1 + B_0 \cdot 2^0 + B_{-1} \cdot 2^{-1} + B_{-2} \cdot 2^{-2} + \cdots B_{-m} \cdot 2^{-m}$$

整数部分转换规则：

- 写出要转换的十进制数
- 写出所有小于此数的二进制各位权值
- 十进制数减去二进制权值，权值由大到小
  - 如够减，相应二进制位记1；
    如不够减，相应二进制位记0
- 不断反复，直到该数为0

小数部分的转换规则：

同整数部分

例： 27D = ? B

27	11	3	3	1
-↓	-↓	-↓	-↓	-↓
16	8	4	2	1
1	1	0	1	1

∴ 27D = 11011B

十进制数转换为二进制数

方法2：除法

$$D = B_{n-1} \cdot 2^{n-1} + B_{n-2} \cdot 2^{n-2} + \cdots + B_1 \cdot 2^1 + B_0 \cdot 2^0 + B_{-1} \cdot 2^{-1} + B_{-2} \cdot 2^{-2} + \cdots B_{-m} \cdot 2^{-m}$$

整数部分转换规则：

- 将十进制整数用基数2连续去除，直到商为0为止；
- 将每次除得的余数反向排列，就可得到十进制数整数部分的转换结果。
- 反向排列是指最后得到的余数排在前边，作为结果的最高位，最先得到的余数排在后边，作为结果的最低位

小数部分的转换规则：

- 将十进制数的小数部分用基数2连续去乘，直到小数部分为0或达到精度为止；
- 将每次所得的乘积的整数部分正向排列，就可得到十进制小数的转换结果。
- 正向排列是指最先得到的整数为结果的最高位，最后得到的整数为结果的最低位

例 N = 117.8125D

$$D = a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \cdots + a_1 \cdot 2^1 + a_0 \cdot 2^0 + b_0 \cdot 2^{-1} + b_1 \cdot 2^{-2} + \cdots b_m \cdot 2^{-m}$$

- 整数部分：117D

117/2=58　　a_0=1

58/2=29　　a_1=0

29/2=14　　a_2=1

14/2=7　　a_3=0

7/2=3　　a_4=1

3/2=1　　a_5=1

1/2=0　　a_6=1

117D = 1110101B

- 小数部分：0.8125D

0.8125×2=1.625　　b_0=1

0.625×2=1.25　　b_1=1

0.25×2=0.5　　b_2=0

0.5×2=1.0　　b_3=1

0.8125D = 0.1101B

N = 117.8125D = 1110101.1101B

(3) 二进制数转换为（八进制）十六进制数

- 将二进制数以小数点为界，向左、向右分别按（3位）4位一组划分，不足（3位）4位的部分用“0”补足（整数部分左补0，小数部分右补0），将每一组数写成一位对应的（八进制）十六进制数，就可得到转换结果。

例如：1110101.11B=？H

0111　0101 . 1100
　7　　5　.　C

∴ 1110101.11B=75.CH

(4) （八进制）十六进制数转换为二进制数

- 将十六进制数以小数点为界，向左、向右分别展开为（3位）4位二进制数，就可得到转换结果。

例如：EA.11H=？B

E	A	.	1	1
1110	1010		0001	0001

∴ EA.11H=11101010.00010001B

1.2　二进制数和十六进制数运算

- 二进制数加法/减法规则

  二进制加法规则：

  0 + 0 = 0

  1 + 0 = 1

  0 + 1 = 1

  1 + 1 = 0——向高位进位为1（逢2进1）

  二进制减法规则：

  0 – 0 = 0

  1 – 0 = 1

  0 – 1 = 1 ——向高位借1（借1当2）

  1 – 1 = 0

- 十六进制数
  - 自学

```
  1 0 1 1 0 1 1 1.1 1
+ 0 0 0 0 0 0 0 1.0 0
-------------------
  1 0 1 1 1 0 0 0.1 1
```

1.3 计算机中数和字符的表示

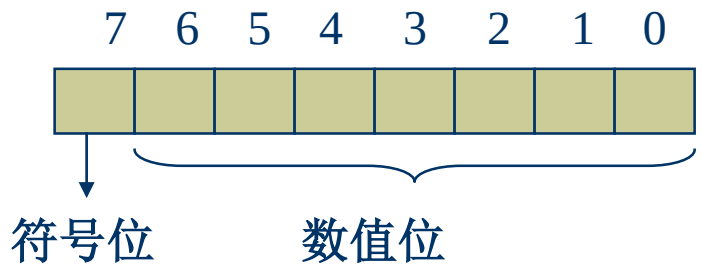
1.3.1 数的补码表示

1.3.2 补码的加法和减法

1.3.3 无符号整数

1.3.4 字符表示法

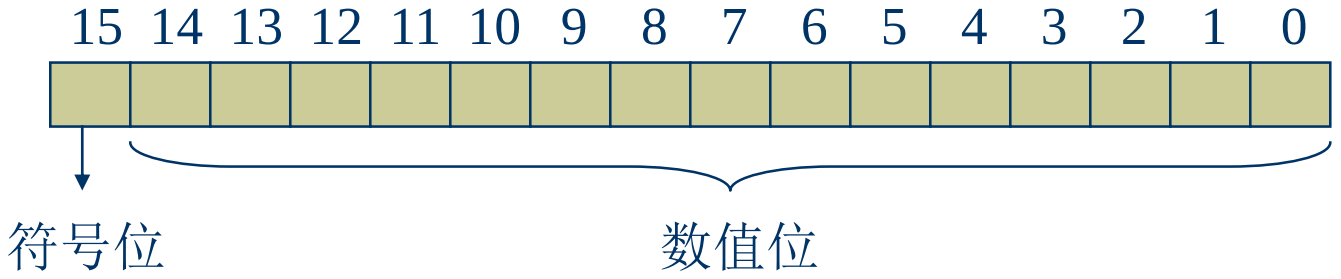
- 数（机器数）的表示：
  - 计算机中的数用二进制表示，数的符号也用二进制表示。
  - 机器数：数连同其符号在内数值化表示。
  - 机器字长：指参与运算的数的基本位数，标志着计算精度，一般是字节的整数倍（8位、16位、32位、64位等）。
  - 假设机器字长n为8位



符号位=0 表示正数

符号位=1 表示负数

  - 假设机器字长n为16位



- 机器数常用表示法 — 原码, 反码, 补码

(Sign-Magnitude, Ones‘ complement, Two's complement)

1.3.1 数的补码表示法

1. 原码表示法

（1）原码表示法： 将数的真值形式中的正（负）号，用代码0(1)来表示，数值部分用二进制来表示。符号 + 绝对值

正数：符号位为0，后面的n-1位为数值部分

负数：符号位为1，后面的n-1位为数值部分

（2）原码的特点

- “0”的原码有两种表示法

  $[+0]_{原}=00000000B$, $[-0]_{原}=10000000B$

- n位二进制原码所能表示的数值范围为： $-(2^{n-1}-1)\sim(2^{n-1}-1)$
- 原码表示一个数时，最高位为符号位

例：n=8bit

$[+3]_{原码}$ = 0 000,0011 = 03H

$[-3]_{原码}$ = 1 000,0011 = 83H

$[+0]_{原码}$ = 0 000,0000 = 00H

$[-0]_{原码}$ = 1 000,0000 = 80H

0的表示不唯一

2. 反码表示法

- 正数的反码同原码，负数的反码数值位与原码相反
- 例：n=8bit

$[+5]_{反码}$ = 0 000,0101 = 05H

$[-5]_{反码}$ = 1 111,1010 = FAH

$[+0]_{反码}$ = 0 000,0000 = 00H

$[-0]_{反码}$ = 1 111,1111 = FFH

0的表示不唯一

3. 补码表示法

（1）补码表示规则：

- 正数的补码: 符号 - 绝对值（与正数的原码相同）

  $[+1]_{补码}$ = 0000 0001 = 01H

  $[+127]_{补码}$ = 0111 1111 = 7FH

  $[+0]_{补码}$ = 0000 0000 = 00H

- 负数的补码: 负数 **X** 用 2^n-|X| 表示

  $[-1]_{补码}$ = 2^8-1 = 1111 1111 = FFH

  $[-127]_{补码}$ = 2^8-127 = 1000 0001 = 81H

  一种简单方法：

  （1）写出与该负数相对应的正数的补码

  （2）按位求反

  （3）末位加1

[-1]补=?

[+1]补= 0000 0001

1111 1110

[-1]补= 1111 1111

例： 机器字长8位，$[-46]_{补码}$ = ？

$[+46]_{补码}$ = 0010 1110

按位求反

1101 0001

末位加1

$[-46]_{补码}$ = 1101 0010 = D2H

补码的符号扩展问题

- 指一个数从位数较少扩展到位数较多时应该注意的问题
  - 8位扩展到16位，16位扩展到32位，等
- 补码表示的扩展规则
  - 正数：前边补0
  - 负数：前边补1

机器字长8位，$[-46]_{补码}$ = 1101 0010B = D2H

机器字长16位，$[-46]_{补码}$ = 1111 1111 1101 0010B = FFD2H

（2）补码的特点

- “0”的补码表示是唯一的

  $[+0]_{补}=[-0]_{补}=00000000B$

- 补码运算时符号位无需单独处理
- 采用补码运算时，减法可用加法来实现

$[13-10]_{补}=[13]_{补}+[-10]_{补}=00001101+11110110=00000011B=[+3]_{补}$

（有进位，自动丢失，符号位为0结果为正）

$[10-13]_{补}=[10]_{补}+[-13]_{补}=00001010+11110011=11111101B=[-3]_{补}$

（无进位，符号位为1，结果为负）

- 符号扩展问题简单

补码的表数范围

n位补码的表数范围： $-2^{n-1} \le N \le 2^{n-1}-1$

n=8　　-128 ≤ N ≤ 127　　2^8个数

n=16　-32768 ≤ N ≤ 32767　　2^{16}个数

比原码和反码多表示一个数，表数效率也高一点

1.3.2 补码的加法和减法

求补运算：对一个二进制数按位求反后在末位加1的运算

补码表示的数具有以下特点：

$$[X]_{补} \xrightarrow{求补} [-X]_{补} \xrightarrow{求补} [X]_{补}$$

$$[117]_{补} \xrightarrow{求补} [-117]_{补} \xrightarrow{求补} [117]_{补}$$

补码加法和减法的规则

$$[x + y]_{补} = [x]_{补} + [y]_{补}$$

$$[x - y]_{补} = [x]_{补} - [y]_{补}$$

$$= [x]_{补} + [-y]_{补}$$

- 补码减法可转换为补码加法
- 不必判断数的正负，符号位一起参加运算，能自动得到正确结果

举例说明补码运算

已知：**X= -27D**

Y= +29D

求：$[X + Y]_{补} = ?$

$[X - Y]_{补} = ?$

解：运算步骤

求$[X + Y]_{补}$：

1）求出$[X]_{补}$，$[Y]_{补}$

$[X]_{补}$ = 11100101B ， $[Y]_{补}$ = 00011101B

2）求出 $[X + Y]_{补}$

$[X + Y]_{补} = [X]_{补} + [Y]_{补}$

= 11100101 + 00011101

= 00000010B

求$[X - Y]_{补}$：

$[X]_{补}$ = 11100101B， [-Y]补 = 11100011B

1）$[X - Y]_{补} = [X]_{补} - [Y]_{补}$

= 11100101 - 00011101

= 11001000B

2）$[X - Y]_{补} = [x]_{补} + [-y]_{补}$

= 11100101 + 11100011

= 11001000B

1.3.3 无符号整数

- 当程序处理的数全是正数时,保留符号位就没有意义了,此时应该采用无符号数表示
- 表数范围

  16位无符号数的表示范围

  $0<=N<=2^{16}-1=65535$

  8位无符号数的表示范围

  $0<=N<=2^{8}-1=255$

- 一般用途：表示地址的数，双精度数的低位字

表数范围

表数范围：给出n位二进制整数的表数范围

（1）无符号整数的范围

8位二进制数所能表示的无符号整数的范围是

0 ~ 255； 2^8=256个数

16位二进制数所能表示的无符号整数的范围是

0 ~ 65535； 2^{16}=65536个数

n位二进制数N能够表示的无符号数的范围

0 ~ (2^n-1)； 2^n个数

表数范围

（2）采用补码表示的带符号整数的表数范围

8位二进制数所能表示的带符号整数的范围是

-128 ~ 127；

16位二进制数所能表示的带符号整数的范围是

-32768 ~ +32767；

n位二进制数N能够表示的带符号数的范围

$-2^{n-1} \sim 2^{n-1}-1$

n位二进制补码的表数范围

十进制	二进制	十六进制	十进制	十六进制
	n=8		n=16	
+127	0111 1111	7F	+32767	7FFF
+126	0111 1110	7E	+32766	7FFE
…	…	…	…	…
+2	0000 0010	02	+2	0002
+1	0000 0001	01	+1	0001
0	0000 0000	00	0	0000
-1	1111 1111	FF	-1	FFFF
-2	1111 1110	FE	-2	FFFE
…	…	…	…	…
-126	1000 0010	82	-32766	8002
-127	1000 0001	81	-32767	8001
-128	1000 0000	80	-32768	8000

BCD 码

- 十进制数在机器中通常采用BCD码表示，而字符及字符串通常用ASCII码表示
- BCD码是一种用二进制编码的十进制数，即用4位二进制形式（0000B-1001B）来表示一位十进制数（0-9），但每4位二进制数之间的进位又是十进制的形式

579 D = 010101111001 BCD

BCD码
- 压缩的BCD码（用4位二进制表示）
- 非压缩的BCD码（用8位二进制表示，前面4位为0000）

1.3.4 字符表示法

- 计算机处理的信息并不全是数,有时需要处理字符或字符串。又因为机器中只能存储二进制数，所以字符在机器里必须用二进制数来表示。一般采用目前最常用的美国信息交换标准代码（ASCII：American Standard Code for Information Interchange）来表示
- ASCII码：用一个字节来表示一个字符，其中7位为字符的ASCII码值，最高位一般用作校验位。

常用字符的ASCII码

字符	ASCII码
0～9	30H~39H
A～Z	41H~5AH
a～z	61H~7AH
回车CR	0DH
换行LF	0AH
$	24H
空格(SPACE)	20H

目前常用输入输出设备显示器、打印机、键盘等均采用ASCII码

1.4 几种基本的逻辑运算

1.4.1 AND “与”运算

1.4.2 OR “或”运算

1.4.3 NOT “非”运算

1.4.4 XOR “异或”运算

逻辑变量: 只能有0或1两种取值

1.4.1 AND “与”运算

A	B	F=A∧B
0	0	0
0	1	0
1	0	0
1	1	1

1.4.2 OR “或”运算

A	B	F=A ∨B
0	0	0
0	1	1
1	0	1
1	1	1

1.4.2 NOT “非”运算

A	$F=\bar{A}$
0	1
1	0

1.4.4 XOR “异或”运算

A	B	F=A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

作 业

P13

1.1, 1.2, 1.3, 1.5, 1.7

注：1.5题的十六进制改为十进制

汇编语言的特点

1. 汇编语言相对于机器指令易于编程、阅读等
2. 汇编语言与机器指令一一对应，可充分理解计算机的操作过程

   有助于理解指令在机器中的执行过程
3. 汇编语言是靠近机器的语言

   可以充分利用机器硬件的全部功能，发挥机器的特点
4. 汇编语言效率高于高级语言

   目标程序的高效率反映在运行速度（时间）、目标代码短（空间）

高级语言

- 高级语言包括面向过程的语言和面向对象的语言，是更接近人类语言（英语）和数学语言的计算机语言。
- 面向过程语言的出现为程序设计带来了方便，要求写出每个任务完成的一系列明确过程。 （ C、 BASIC、 Pascal、FORTRAN）
  - 一般按指令顺序执行
- 面向对象语言是从面向过程语言发展而来的，它改变了编程者的思维方式，采用与人们认识世界相同的方法，将问题分解成若干对象和对象间的相互作用。 （c++、java ）
  - 可乱序执行，只要资源或条件满足就可执行，提高资源利用率和程序效率
  - 更适合现代计算机中的多CPU结构和多线程技术等
  - 使程序设计更接近于自然语言，设计出来的软件质量更高

汇编语言与高级语言的比较

1. 程序特点

高级语言－面向问题：：汇编语言－面向机器

2. 软件开发

高级语言：节省开发时间，但不允许程序员直接使用微处理器的许多特性（寄存器、标志、内存等）

汇编语言：编程比较难，但可充分发挥机器特性的作用

3. 代码生成

高级语言：编译后目标代码程序比较长，效率低

汇编语言：目标代码程序短，执行速度快，占用内存少